Validation of a System Design Framework with Formal RDF Techniques

The CCC design framework

Michael Dossis*[1]

Department of Informatics and Computer Technology, TEI of Western Macedonia
Kastoria Campus, Fourka Area, Kastoria, GR 52 100, Greece

*[1]mdossis@yahoo.gr

Abstract

The continuously increasing complexity of contemporary embedded systems and portable computing machines has motivated research in high-level and system synthesis (HLS). In this work, the HLS internal formats and transformations that are applied through compilation, on the source code are formal, therefore the generated system implementation is correct-by-construction. The use of RDF triples for making statements about Web resources, is extended in this work for the formal definition of the internal object relations of a hardware compiler. A prototype HLS tool which has been developed by the author is analyzed here. This toolset uses compiler-generators, RDF rules and logic programming in combination with XML validation of the internal state of the compiler. All of these formal relations make the compilation process formal. The targeted implementation hardware models are free from any technology or vendor specifics, so the prototype tool can be connected into any industrial or academic system development flow. The HLS tool is enhanced with the Parallel, Abstract Resource–Constrained Scheduler, which aggressively optimizes the initial state schedules, into maximally parallelized ones. A number of test cases from real-world applications prove the usability of the embedded scheduler PARCS and the formal compilation of the design flow.

Keywords

High Level Synthesis; Formal Languages; Electronic Design Automation; Scheduling; Compilers; RDF; Formal Methods; Logic Programming

Introduction

Digital systems integrated in embedded and portable products, have highly complex components, design hierarchy and interconnections. Industry and academia have invested in new HLS methodologies, so as to improve automation, quality of implementations and specification-to-product times. Nevertheless, HLS tools are still not widely accepted by the engineering community because of their poor results, especially for large applications with complex module and control-flow hierarchy. An example of such a prohibitive factor is that quite often, the programming style of the source code has a severe impact on the quality of the synthesized implementation. For such complex applications, the complexity of the synthesis transformations (front-end compilation, algorithmic transformations, optimizing scheduling, allocation and binding), increases exponentially, when design size increases linearly.

Most of the existing commercial or academic HLS tools impose proprietary extensions or restrictions on the programming model of the specifications that they accept as input, and various heuristics on the HLS transformations that they utilize. The great majority of such tools is suitable for linear, dataflow dominated (e.g. stream-based) designs, such as pipelined DSP, image processing and streaming.

The contribution of this work is a unified design framework in which an integrated set of HLS transformations is utilized that are based on formal relations, thus the produced hardware implementations are correct-by-construction. This means that due to the formal nature of the HLS tasks, the functionality of the resulting implementation matches the functionality of the source code. Therefore, the design needs verification only at the top behavioral level, without spending days or even weeks, on lengthy RTL or annotated gate simulations. Another achievement of this methodology is that it maintains the hierarchy boundaries and object (data and operational) names of the design, from the original algorithmic description down to the hardware implementation (e.g. the generated VHDL RTL). In this way, the generated RTL is fully readable and its names can be traced back to the source program model.

The HLS design framework that the author of this work has designed and developed (Dossis, 2011), includes a scheduler of operations into control steps, or states, achieving the maximum functional parallelism in the synthesized implementation (Paulin, 1989). This HLS scheduler called PARCS, utilizes logic programming (Nilsson, 1995) and RDF subject-predicate-object relations (Allemang, 2011) to achieve maximum operation parallelism in a formal manner. Thus, the functional correctness of the delivered implementations is guaranteed. Scheduling algorithms have been explored in (Walker, 1995). The author's prototype HLS framework utilizes RDF relations and logic programming to deliver high quality implementations with code readability and an admirable controllability of the implementation hierarchy.

The next section presents current and previews work on HLS and RDF technology. In section 3, the author's prototype hardware compilation framework is outlined, with the use of formal predicate logic, RDF and XML validation. Section 4 analyzes some testcases used to verify the usability and correctness of the CCC HLS toolset, and produce control-state statistics of the optimized schedules. The last section draws useful conclusions from this work and proposes future extensions on the developed HLS design methodology.

Previous Work in Formal HLS and RDF/XML Methods

HLS utilizes the tasks scheduling, allocation and binding which have been studied extensively for more than two decades, in the bibliography (Walker, 1995). At least, the front-end of HLS tools have often learnt techniques from software programming language compilers (Holub, 1990). An example of successful intermediate format is the Electronic Design Interchange Format (EDIF) (Wikipedia, 2012), (Rubin, 2012), used by most E-CAD tools. Optimizing algorithms that can be applied in complex control flow designs have been analyzed in (Kountouris, 2002), (Gupta, 2004), (Wang, 2003).

A HLS approach tailored to the design of distributed logic and memory architectures was explained in (Huang, 2007). Communicating processes are part of a system specification which is implemented with the method of (Huang, 2003). Memory access management is incorporated in a HLS design flow in (Gal, 2008). It mainly targets digital signal processing (DSP) applications as well as streaming systems in general using specific performance constraints. Mutually exclusive scheduling on the extended data-flow graph (EDFG), based on the signal flow graph, was explained in (Gupta, 2003). The HLS approach in (Molina, 2009) takes as input a behavioral design description and time constraints. Then it selectively decomposes complex operations into smaller ones, so as to schedule in every clock cycle, a similar set of decomposed fragments of operators, with the same pattern.

Actors are used in (Keinert, 2009) to model every module or process of a system model. Every actor communicates with other actors via a number of communication channels and they are both used to build system model in (Keinert, 2009). The SystemCoDesigner (Keinert, 2009) used these actors to integrate HLS into electronic system level (ESL) design space exploration. The SURYA validating system (Kundu, 2010) applied the Symplify theorem prover. SURYA attempts to prove that every HLS translation of a specification (code) model generates a RTL model that is functionally-equivalent to the one in the behavioral input. SURYA was used to validate the SPARK HLS tool (Gupta, 2004), and it found two bugs in the SPARK compilations which were unknown before. Replacing flip-flop registers with latches is used in (Paik, 2010) to produce better timing in the implementations, since latches are inherently more tolerant to process variations than flip-flops. Nevertheless, latches are difficult to characterize and handle, particularly in complex designs, as compared to flip-flops.

The W3C Resource Description Framework

The Resource Description Framework (RDF), a metadata data model, is used to model the information of web resources. The web information exists in a variety of formats and syntactic variances (Allemang, 2011). RDF descriptions utilize subject-predicate-object relations which are based on the idea of making explicit statements about various resources, and in the RDF terminology, they are called triples. The RDF model is extensively used for knowledge and information storage and retrieval for large automated software packages which are not all related to Web information models. This is because of the RDF's suitability to capture, store, exchange, and use machine-readable information distributed throughout the Web. In this work, RDF is used to formally represent internal knowledge and state in the author's prototype HLS design framework and compiler.

One of the W3C specifications that were used to define

RDF is the eXtensible Markup Language (XML) serialization format. XML's syntax is formally defined (Allemang, 2011), and it is very suitable for modelling simple triples such as subject-predicate-object (and other) formal relations. As an example the following RDF code:

```
<rdf:RDFxmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description    rdf:about="http://www.awl.com/Formal_Synthesis">
<dc:title>Formal Synthesis</dc:title>
<dc:publisher> Addison-Wesley Publishing</dc:publisher>
</rdf:Description>
</rdf:RDF>
```

is the RDF representation of the following fact: "The title of this resource, published by the Addison-Wesley Publishing company, is Formal Synthesis". RDF and XML are formal means to define object relations that may also represent knowledge about data in large programs, and can be used to validate internal data forms such as those of E-CAD tools. XML files are easily readable for both humans and machines. An XML schema is a formal definition of an XML document type, and contains constraints about the content and structure of documents which can be used for the validation of a certain document instance. Well-known formalisms of the XML schema are the document type definition (DTD) language, the XML Schema and the Relax NG formats (Allemang, 2011).

A valid XML document instance can be validated against the rules of an XML schema. This is done by certain XML parsers which are compatible with specific XML schema implementations such as the DTD or the Relax NG languages. The following example of an XML schema instantiation is automatically produced by the author's prototype front-end compiler (see following paragraphs), and it contains the definition of a data type and two subprograms:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Produced by CCC front-end compiler -->
<schemaxmlns=http://www.w3.org/2001/XMLSchema
targetNamespace="http://www.w3.org/2001/XMLSchema">
<annotation>
<documentation>
XML schema for a hierarchical module of the source code
</documentation>
</annotation>
<complexType name=" hierarchy_part">
<sequence>
<element name=" type_def_natural_2048"/>
<sequence>
<element name=" data_object_variable2 "/>
<element name=" data_object_constant1_value_100 "/>
<element name=" data_object_constant1_value_1000 "/>
 </sequence>
<element name=" function_convert3"/>
<sequence>
<element name=" input_parameter_my_input1 "/>
<element name=" input_parameter_my_input2 "/>
<element name=" input_parameter_my_input3 "/>
</sequence>
<element name=" procedure_differential2"/>
<sequence>
<element name=" input_parameter_my_input4 "/>
<element name=" input_parameter_my_input5 "/>
<element name=" output_parameter_my_output1 "/>
</sequence>
</sequence>
</complexType>
</schema>
```

The above XML instance is produced automatically from the internal data of the front-end compiler, and it defines the data type "natural_2048" and two subprograms, the function "convert3" with three formal input parameters "my_input1", "my_input2" and "my_input3" as well as procedure (see ADA-95 definition) differential2 with two formal input parameters and one output.

All of these three top-level elements are of type "hierarchy_part", which is a specific data type for this XML schema. Each one of them, include a sequence of "leaf" sequences which contain the structure of all child elements with type which is defined above.

The graphical validation result of the above XML code shown in Fig. 1 was automatically produced by a DTD parser and helped to initially identify and correct afterwards a number of semantic differences in the source code of the HLS tool.

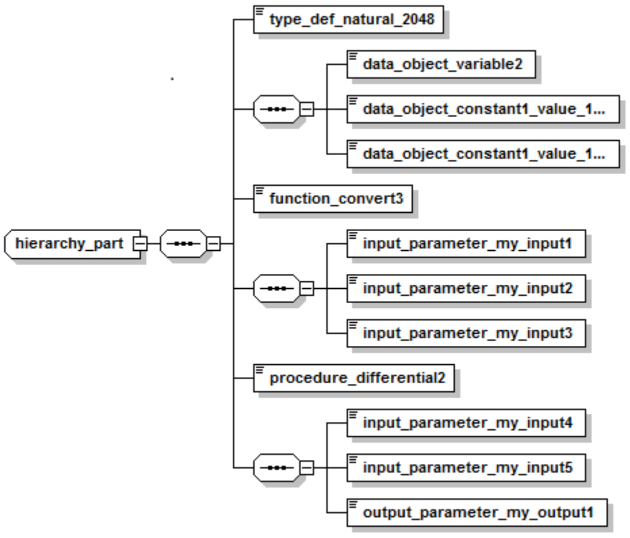
type_def_natural_2048
data_object_variable2
data_object_constant1_value_1...
data_object_constant1_value_1...
function_convert3
hierarchy_part
input_parameter_my_input1
input_parameter_my_input2
input_parameter_my_input3
procedure_differential2
input_parameter_my_input4
input_parameter_my_input5
output_parameter_my_output1

FIG. 1 GRAPHICAL VALIDATION OF AN XML DOCUMENT INSTANCE AGAINST A SPECIFIC XML SCHEMA (AUTOMATICALLY PRODUCED FROM INTERNAL COMPILER DATA)

The use of XML as a formal way to model internal information in design (E-CAD) tools has been reported in research approaches. XML representation of C code has been introduced in (Atsumi, 2011) to propel the development of CASE tools. A hash table and a stream index were used in (Weijian, 2011) to filter out invalid elements of an XML document. The IEEE std 1685-2009 IP_XACT, based on XML schema, is used as internal design representation and exchange between EDA tools (IEEE IP-XACT std, 2010). This information can be design units, interconnections, functional primitives, metadata, and IP blocks.

XML schema and the eXtensible Stylesheet Language Transformations (XSLT) language have been used in (Jumaa, 2010) to facilitate the sharing, conversion, transfer and exchanging of healthcare database information. A large number of academic and commercial database and web processing systems, use XML schemas and instantiations to represent internal data and information representations that are either primary data or generated automatically or semi-automatically by information processing routines, found database, multi-media and web processing systems.

The system in (Sanchez-Martinez, 2010) combined XML with some Web technologies to structure, consulte and share corporate data. An XML document similarity algorithm was presented in (Sun, 2010). The existing query matching and compensation framework in IBM DB2 is enhanced with new functionality, in order to implement query rewrite rules that enable the use of various views in the evaluation of queries over XML in (Godfrey, 2009). An incremental approach of XML-to-relational mapping storage, named "T-Schema" was proposed in (Xu, 2010), in order to address the strong dynamics of XML. The SMOQE system (Fan, 2007) implemented the first regular XPath engine and provided a solution to answering queries over recursively defined XML views. The opportunities and challenges for utilizing semantic web technologies for EIS and databases have been discussed in (Rajugan, 2006). A metamodel (Kassab, 2008) was used to capture NFRs and their relations, using a XML implementation.

The Intermediate Predicate Format

The Intermediate Predicate Format (IPF) [1] was developed by the author of this paper as a method for the design encapsulation and the HLS transformations in the CCC HLS design framework [2]. A near-complete analysis of IPF syntax and semantics can be found in (Dossis, 2010). IPF utilizes the resolution of Horn clauses (Nilsson, 1995), as the building blocks of a HLS tool with formal transformations (e.g. state optimization engine). The IPF database produced by the front-end phase of the CCC compiler (see following paragraphs), captures the complete algorithmic, structural and data typing information of the source programs. The IPF syntax is as follows:

fact_id(object1, object2, …, objectN) (form 1)

The predicate name fact_id relates the objects object1 to objectN in a logical way. The identifier fact_id gives the name of the logical relation between objects. IPF facts can represent a program operation, a data object description, a data type, an operator, a subprogram call, etc. These logical relations of objects, are used by the back-end phase of the CCC compiler in order to implement the HLS transformations of the tool. The CCC HLS transformations use the IPF facts, along with the internal logical rules in order to “conclude” and generate the RTL (register-transfer level) hardware models. Due to its syntax, IPF facilitates declarative processing within Prolog code, as well as linear, sequential processing. The latter is guranteed in the way that some IPF facts (e.g. data table facts) are referenced with their linear entry numbers in other IPF

[1] The intermediate predicate format is patented with patent number: 1006354, 15/4/2009, from the Greek Industrial Property Organization

[2] This hardware compiler method is patented with patent number: 1005308, 5/10/2006, from the Greek Industrial Property Organization

facts (e.g. program table facts).

Another view of the IPF's statements is the XML schema view which is an XML instance of the internal state of the front-end and back-end phases of the prototype HLS hardware compiler. The program statement (Dossis, 2010) according to form 1 is given in XML instance as in the following code excerpt:

```
<complexType name=" prog_stmt">
<sequence>
<element name=" module_subprogram2"/>
<element name=" entry_number_3"/>
<element name=" states_0"/>
<element name=" operator_63"/>
<element name=" left_operand_dx"/>
<element name=" right_operand_dy"/>
<element name=" result_operand_xc"/>
<element name=" next_operation_5"/>
</sequence>
</complexType>
```

The graphical validation of the above program statement XML instance is shown in Fig. 2.

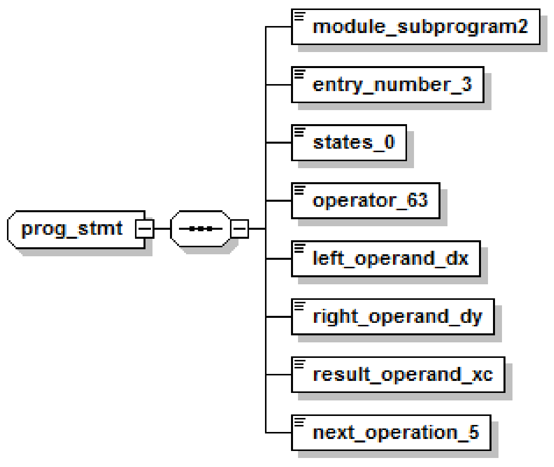

FIG. 2 GRAPHICAL VALIDATION OF THE PROGRAM STATEMENT XML INSTANCE.

The CCC HLS Hardware Compiler

The CCC Design Flow

The hardware compilation flow consists of two major steps: the front-end phase and the back-end phase. These two stages are exchange data through IPF. The tool generates a functionally-equivalent XML schema instance of IPF in order to validate the compilation process, using graphical or command-line based IPF validators. The front-end phase executes typical tasks of a program compiler, such as lexical and syntactic analysis, generation of syntactic and semantic error messages, production and processing of abstract syntax trees, type-checking of data and program statements, optimization of auxiliary variables and constants etc.

Using the XML formal view of the IPF model, the XML schema view, which is also produced automatically by the front-end and back-end phases to formally validate the translation, IPF constitutes a formal link between the two phases of the hardware compilation.

The initial FSM state schedules are directly produced via analyzing the IPF data, as well as additional user parameters (such as e.g. the use of shared memories for large data objects). Then, and during the back-end phase, these schedules are optimized using a parallelizing transformation algorithm called PARCS (Parallel Abstract Resoure – Constrained Scheduling). PARCS works with or without user–provided resource constraints, and delivers the optimal parallel (and so faster) hardware implementation, while satisfying data and control dependencies in the source programs.

Formal Back-end Compiler Transformations

The inference engine rules of the back-end compiler are implemented with definite clauses (Nilsson, 1995) of the following form:

$A0 \leftarrow A1 \wedge \ldots \wedge An$ (where $n \geq 0$) (form 2)

where $\leftarrow$ is the logical implication symbol ($A \leftarrow B$ means that if B applies then A applies), $\wedge$ is the logical conjunction symbol, and A0, …, An are atomic formulas (logic facts) of the form:

predicate_symbol(Var_1, Var_2, …, Var_N) (form 3)

where the positional parameters Var_1,…,Var_N of the above predicate "predicate_symbol" are either variable names (in the case of the back-end inference rules), or constants (in the case of the IPF table statements) (Nilsson, 1995). Therefore, the input programs are formally transformed into a set of provably-correct hardware accelerators.

Formal validation using XML schema

Formal logic rules (logic relations) as in form 2 are used to build the back-end inference engine. A new design to be synthesized is loaded via its IPF view into the back-end inference engine. Hence, the IPF's statements "drive" the logic rules of the back-end compiler which generate provably-correct hardware implementations. The hardware compiler generates hardware models which are technology-independent, free of any standard template, and custom microarchitectures. These models are generated as synthesizable VHDL/Verilog.

As mentioned above, the PARCS optimizer processes either the initial or the enhanced schedule, depending on user options. The enhanced schedule contains additional communication hardware protocols which include automatically generated external memory ports and interfaces that the user may want to implement (for external, shared memories). XML schema is used to validate the hardware compilation intermediate representations and processes, at the various stages of the hardware compiler. The following code is part of the XML schema validation of the state(…) predicate inference rule:

```
<complexType name="state">
<sequence>
<element name="Module_1"/>
<element name="parcs"/>
<element name="parcs_state_number"/>
<element name="parcs_state_name"/>
<element name="parcs_next_state"/>
<element name="no_conditional_transition"/>
<element name="scheduled_operation_list">
<complexType>
<sequence>
<element name="Operation_1"/>
<element name="Operation_2"/>
<element name="Operation_3"/>
</sequence>
</complexType>
</element>
<element name="no_conditional_operations"/>
</sequence>
 </complexType>
```

This XML instance models a PARCS state of design Module_1, with three scheduled operations in parallel and with no conditional operations or transitions. Both the logic and the XML views of the IPF which are produced automatically by the front-end compilation phase as well as the back-end inference rules are modelled and validated in both logic programming and XML code. The graphical validation of the above XML PARCS state code is shown in Fig. 3.

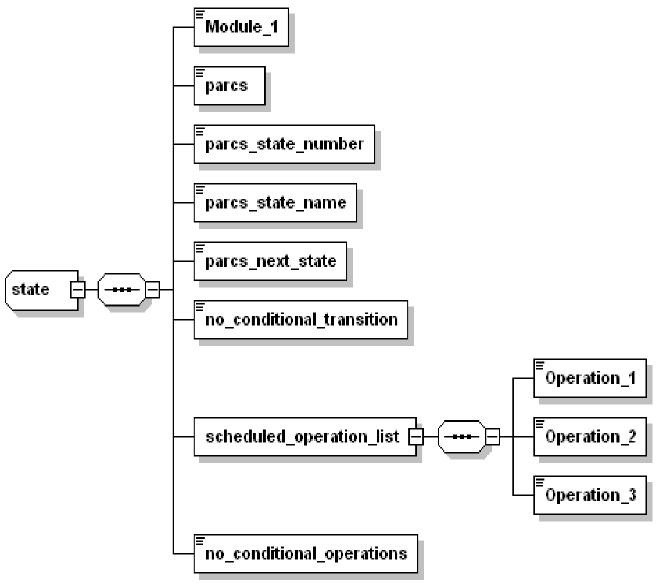

FIG. 3 XML SCHEMA VALIDATION OF A PARCS STATE

The XML view of the internal transformations and data of the back-end compiler contains relations between predicate symbols, e.g. transform1 (HLS transformation), and algorithmic items, e.g. operation1, operand_A, operand_B, result_C. An example of logic programming view of this relation is the following:

transform1(operation1, operand_A, operand_B, result_C).

The XML schema view of this relation is shown in the following part of XML code:

```
<complexType name=" transform1">
<sequence>
<element name=" operation1"/>
<element name=" left_operand_A "/>
<element name=" right_operand_B "/>
<element name=" result_data_C "/>
</sequence>
</complexType>
```

The XML validation of the above transformation predicate in XML code, is given in a graphical manner in Fig. 4.

All of the generated XML schema instances are validated using graphical or text validators available on the web. As it can be seen from the above logic and XML views that the XML view is much more verbose than the Prolog code, so it is used exclusively to implement the formal validation of the compiler's intermediate forms and transformations.

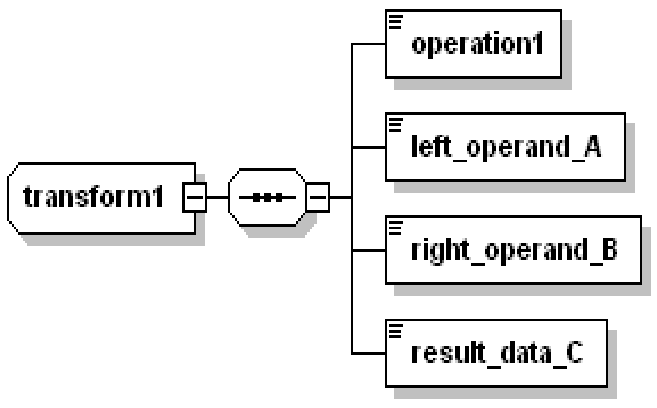

FIG. 4 XML SCHEMA VALIDATION OF A BACK-END COMPILER TRANSFORMATION

Experimental Results and Statistics

Five such ADA designs executed with the CCC framework are outlined here. The five benchmarks include a DSP FIR filter, a second order differential equation iterative solver, a RSA crypto-processor from cryptography applications, an application that uses two level nested for-loops, and a large MPEG video compression engine.

All of the above hardware models were simulated, and the RTL models confirmed the input source program's functionality, as it was expected, due to the formal nature of the hardware compilations. Moreover, for all tests, the intermediate form and the internal HLS transformations were validated against the respective XML schemas which were automatically extracted from critical points in the compilation flow. For example, as mentioned and explained earlier, the XML view of the IPF as well as the optimizing transformations were validated with a graphical XML validation tool, and therefore their formal nature was confirmed.

The input ADA programs model the software and hardware parts of the intended system. The software part of the system is compiled with the GNU ADA tools. The hardware part is rapidly developed with the CCC tools. The experimental environment for execution of the coprocessors is shown in Fig. 5.

The whole system is modelled in the ADA programming language. The system is coded in ADA, compiled with the GNU ADA and the generated binaries are verified with the ADA testbench and test vectors. Then the part of ADA subroutines that are intended for hardware implementation (coprocessors) are isolated and placed in an autonomous ADA package (library module). The latter is compiled and synthesized into hardware implementations using the prototype CCC tools that were built by the author of this paper. The generated hardware modules (coprocesors) are downloaded in a Xilinx Virtex-2 FPGA and the whole system is executed with the host processor as shown in Fig. 5.

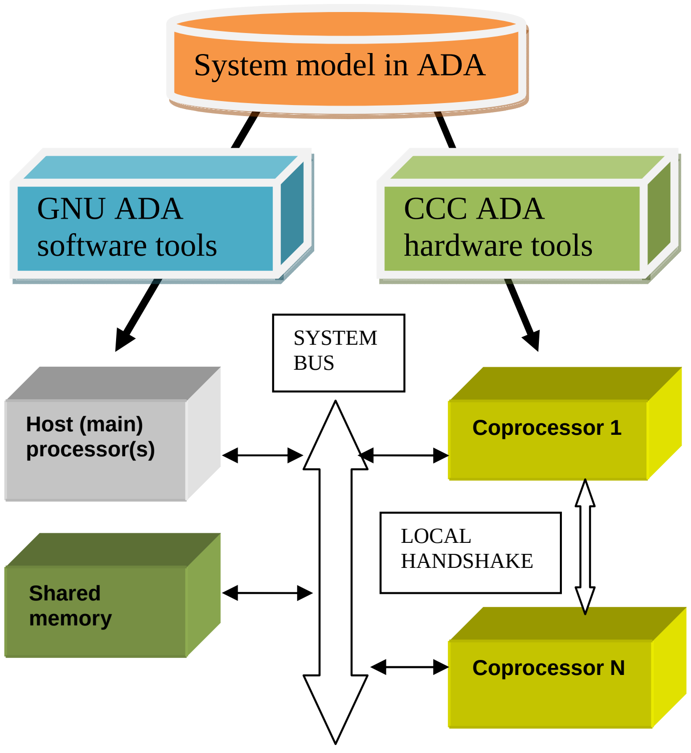

FIG. 5 EXPERIMENTAL DESIGN AND EXECUTION ENVIRONMENT FOR THE GENERATED COPROCESSORS

The generated coprocessors are optimized with the scheduler of the CCC design framework. First, initial operation schedules are produced automatically from the source ADA programs, then enhanced with hardware-specific features and optimized with the PARCS scheduler of the CCC backend HLS compiler. The state reduction statistics after the application of the PARCS optimizing scheduler on the initial schedules, produced from the five benchmark tests, discussed in this section, are listed in Table 1.

From these statistics, it can be seen that the states reduction percentage is climbing up to 41%. This implies particularly efficient implementation for large designs with a few hundred states per module, which is not unusual at all in the case of complex ASICs and complex IP blocks of embedded systems and SoCs.

TABLE 1 STATE REDUCTION STATISTICS FROM THE PARCS OPTIMIZER

Module Name	Initial Schedule States	Optimized (PARCS) States	State Reduction Rate
FIR filter main routine	17	10	41%
Differential equation solver	20	13	35%
RSA main routine	16	11	31%
nested loops 1st routine	28	20	29%
nested loops 2nd routine			
(with embedded memory)	36	26	28%
nested loops 2nd routine			
(with external memory)	96	79	18%
nested loops 3rd routine	15	10	33%
nested loops 4th routine	18	12	33%
nested loops 5th routine	17	13	24%
MPEG 1st routine	88	56	36%
MPEG 2nd routine	88	56	36%
MPEG 3rd routine	37	25	32%
MPEG top routine (with embedded memory)	326	223	32%
MPEG top routine (with external memory)	462	343	26%

The input ADA model subroutines can be standalone or hierarchical. The latter implies that the input code hierarchy is maintained through the CCC compilation. Therefore, the CCC designer can dictate the system modularity of the generated hardware coprocessors. Moreover, all the necessary system interfaces and (e.g. memory) communication protocols are automatically generated by the CCC framework and inserted in the initial operation schedules that are automatically extracted from the source code. In addition to that, an arbitrary number (indicated by the designer) of input ADA subroutines can be “marked” to be implemented in custom arithmetic modules which are inserted as expanded function calls in the generated FSM states, and therefore, executed in one clock cycle. This option was successfully applied on the lower-level subroutines of the FIR DSP filter and RSA crypto-processor designs.

As it can be found in the statistics of Table 1, the number of hardware states are increasingly high, in complex designs as the MPEG engine. In this manner, the contribution of the CCC framework is invaluable, in order to control the development of highly complex designs such as the MPEG coder. It is widely known among the experienced hardware designers that development and verification of highly complex FSMs with over 20-30 states is extremely cumbersome and prune to errors. Nevertheless, with the use of the CCC design framework, development of such complex designs with the aid of ADA compilation and testbenches is becoming a routine of a few hours development time for experienced programmers. More complex ASIC or FPGA designs take usually more than 6 months development and verification time, however, the turn-around time is reduced to a few days with the use of the CCC tools, which results in provably-correct implementations, since formal logic programming and RDF validation techniques are employed.

Conclusions and Future Work

The main contribution of this work is a formal, high-level hardware synthesis method and a unified prototype design framework developed by the author of this paper. The CCC HLS tool utilizes compiler-compiler and logic inference techniques, which makes the hardware compilation process formal. The hardware compilation is enhanced with XML schema validation as well as RDF logic relations for the execution and formal validation of the prototype hardware compiler. In this way, the XML views of the internal format as well as the formal transformations of the CCC compiler are validated and their output is formally verified.

The CCC tools transform a number of arbitrary and general input subprograms (for now coded in the ADA language) into a corresponding number of functionally-equivalent RTL VHDL hardware implementations. A large number of input applications were run through the hardware compiler, five of which were evaluated in this paper. In all cases, the functionality of the produced hardware accelerators matched that of the input subprograms. Encouraging state-reduction rates of the PARCS scheduler-optimizer were observed for five benchmarks in this paper, which exceed 36% in some cases (see Table 1).

Future extensions of this work include on-going upgrading the front-end phase to accommodate more input programming languages (e.g. ANSI-C, C++) and a more globalized use of RDF techniques throughout the flow of the HLS tool. Another extension will be the inclusion of more than two operand operations as well as multi-cycle arithmetic unit modules, such as multi-cycle operators to be used for pipelining. Furthermore, connection of the front-end phase with even more diagrammatic system modelling formats such as the UML design techniques is currently investigated.

REFERENCES

Allemang, D., and Hendler, J. Semantic Web for the Working Ontologist. Second Edition: Effective Modeling in RDFS and OWL, Morgan Kaufmann Publishers, Elsevier Inc., ISBN-10: 0123859654, 2011.

Atsumi, N., Kobayashi, T., Yamamoto, S., and Agusa, K. "An XML C Source Code Interchange Format for CASE Tools". In Proceedings of the IEEE 35th Annual Computer Software and Applications Conference (COMPSAC):498-503, Munich, 18-22 July 2011.

Dossis, Michael. "Intermediate Predicate Format for design automation tools". Journal of Next Generation Information Technology (JNIT), 1(1):100-117, May 2010.

Dossis, Michael F. "A Formal Design Framework to Generate Coprocessors with Implementation Options". International Journal of Research and Reviews in Computer Science (IJRRCS), August 2011, ISSN: 2079-2557, Science Academy Publisher, United Kingdom, www.sciacademypublisher.com, 2(4):929-936, 2011.

Gal, B. L., Casseau, E., and Huet, S. "Dynamic Memory Access Management for High-Performance DSP Applications Using High-Level Synthesis". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16(11):1454-1464, 2008.

Godfrey, P., Gryz, J., Hoppe, A., Ma, W., and Zuzarte, C. "Query Rewrites with Views for XML in DB2". In Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE '09):1339-1350, Shanghai, China, March 29-April 2 2009, ISBN: 978-1-4244-3422-0.

Gupta, Sumit, Gupta, Rajesh Kumar, Dutt, Nikil D., and Nikolau, Alexandru. "Dynamically increasing the scope of code motions during the high-level synthesis of digital circuits". In Proceedings of the IEEE Conference on Computers and Digital Technologies, 150(5):330–337, 2003.

Gupta, Sumit, Gupta, Rajesh Kumar, Dutt, Nikil D., and Nikolau, Alexandru. "Coordinated Parallelizing Compiler Optimizations and High-Level Synthesis". ACM Transactions on Design Automation of Electronic Systems, 9(4):441–470, 2004.

Holub, A. Compiler Design in C. Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA, 1990.

Huang, Weidong, Raghunathan, Anand, Jha, Niraj K., and Dey, Sujit. "High-level Synthesis of Multi-process Behavioral Descriptions". in Proceedings of the 16th IEEE International Conference on VLSI Design (VLSI'03):467-473, 2003.

Huang, C., Ravi, S., Raghunathan, A., and Jha, N. K. "Generation of Heterogeneous Distributed Architectures for Memory-Intensive Applications Through High-Level Synthesis". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15(11):1191-1204, 2007.

IEEE Computer Society and IEEE Standards Association Corporate Advisory Group, IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows, IEEE std 1685-2009, ISBN 978-0-7381-6159-4, IEEE, New York, 18 February 2010.

Jumaa, H., Rubel, P., and Fayn, J. "An XML-based framework for automating data exchange in healthcare". In Proceedings of the 12th IEEE International Conference on e-Health Networking Applications and Services (Healthcom), ISBN: 978-1-4244-6374-9, IEEE:264-269, Lyon, France, 1-3 July 2010.

Kassab, M., Ormandjieva, O., and Daneva, M. "A Traceability Metamodel for Change Management of Non-Functional Requirements". In Proceedings of the Sixth International Conference on Software Engineering Research, Management and Applications, IEEE: 245-254, 2008.

Keinert, Joachim, Streubuhr, Martin, Schlichter, Thomas, Falk, Joachim, Gladigau, Jens, et. al. "SystemCoDesigner—an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications". ACM Transactions on Design Automation of Electronic Systems (TODAES), 14(1), Article No.: 1, 2009.

Kountouris, Apostolos A., and Wolinski, Christophe. "Efficient Scheduling of Conditional Behaviors for High-Level Synthesis". ACM Transactions on Design

Automation of Electronic Systems, 7(3):380–412, 2002.

Kundu, S., Lerner, S., and Gupta, R. K. "Translation Validation of High-Level Synthesis". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29(4):566-579, 2010.

Molina, M. C., Ruiz-Sautua, R., Garcia-Repetto, P., and Hermida, R. "Frequent-Pattern-Guided Multilevel Decomposition of Behavioral Specifications" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28(1):60-73, 2009.

Nilsson, U., and Maluszynski, J. Logic Programming and Prolog. John Wiley & Sons Ltd., 2nd Edition, 1995.

Paik, Seungwhun, Shin, Insup, Kim, Taewhan, and Shin, Youngsoo. "HLS-l: A High-Level Synthesis framework for latch-based architectures" IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29(5):657-670, 2010.

Paulin, P. G., and Knight, J. P. "Force-directed scheduling for the behavioral synthesis of ASICs". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 8(6):661–679, December 1989.

Rajugan, R., Chang, Elizabeth, Feng, Ling, and Dillon, Tharam S. "Modeling Dynamic Properties in the Layered View Model for XML Using XSemantic Nets". Advanced Web and Network Technologies, in Lecture Notes in Computer Science, (3842/2006):142-147, 2006.

Rubin, M. S. Computer Aids for VLSI Design. Appendix D: Electronic Design Interchange Format, 1994. http://www.rulabinsky.com/cavd/text/chapd.html Accessed 9 February 2012.

Sanchez-Martinez, L. D., and Medina-Ramirez, R. C. "An XML information management: a research team case". In Proceedings of the 20th International Conference on Electronics, Communications and Computer (CONIELECOMP):197-200, Cholula, Mexico, 22-24 February 2010, ISBN: 978-1-4244-5352-8.

Sun, X., Cheng, H., and Wang, X. "An XML schema-based similarity algorithm". In Proceedings of the 2nd International Conference on Future Computer and Communication (ICFCC), 1:36-38, Wuhan, China, 21-24 May 2010, ISBN: 978-1-4244-5821-9.

The Electronic Design Interchange Format. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/EDIF Accessed 9 February 2012.

Walker, R. A., and Chaudhuri, S. "Introduction to the scheduling problem", IEEE Design & Test of Computers, 12(2): 60–69, 1995.

Wang, W., Tan, T. K., Luo, J., Fei, Y., Shang, L., et al. "A comprehensive high-level synthesis system for control-flow intensive behaviors". in Proceedings of the 13th ACM Great Lakes symposium on VLSI (GLSVLSI '03):11-14, 2003.

Weijian, Xia, Heji, Zhao, and Jiasheng, Zhao. "The XML filtration based on hash table and stream index". In Proceedings of the International Conference on Mechatronic Science, Electric Engineering and Computer (MEC):1286-1290, Jilin, China, August 19-22, 2011.

Wenfei, Fan, Geerts, F., Xibei, Jia, and Kementsietsidis, A. "Rewriting Regular XPath Queries on XML Views". In Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE 2007):666-675, Istanbul, Turkey, 15-20 April 2007, ISBN: 1-4244-0803-2.

Xu, Yufeng, Ma, Shilong, Yi, Shengwei, and Yan, Yizhou. "From XML Schema to Relations: A Incremental Approach to XML Storage". In Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering (CiSE):1-4, Wuhan, China, 10-12 December 2010, ISBN: 978-1-4244-5391-7.

Michael F. Dossis has an Advanced Engineering Diploma from NTUA, Athens, Greece and a Ph.D. in Electrical and Electronic Engineering from the Un. of Bradford, UK. He is currently an Associate Professor of Informatics and Computer Technology with the Higher TEI of Western Macedonia, Greece. He has been post-doctoral research and teaching staff with the Universities of Bradford and Oxford. (both in UK), as well with rich industrial experience in design and development of multimillion-gate ASICs, FPGAs and processor core designs at LSI Logic, ARM, Virata (now Connexant) and Intracom Telecom (Greece).

During his more than 17-year career in industry and academia, he has developed some millions of lines of high-level program and computer description code. His research interests include design automation, methodologies and tools for the design of digital electronic systems, architectures of application-specific integrated circuits, computer architecture, computer languages and their compilers, high-level synthesis and hardware-software codesign, formal design methods, applications of artificial intelligence, embedded systems, custom processor and computer architectures, and reconfigurable computing. He holds a number of international patents and publications in Formal Methods for hardware and system design-automation and he has developed high-level synthesis and associated tools.

Prof. Dossis is a member of the Technical Chamber of Greece, and of the technical committee of electronic systems of the same organisation. He is also chief editor, associate editor and reviewer in a number of high-quality international scientific journals and conferences.